OsloMet - Oslo Metropolitan University, Department of Computer Science, Norway

ACIT4420-1 24H Problem-solving with scripting, Master in ACIT, 2024

Final Assignment

Shailendra Bhandari & Rabindra Khadka

October 19, 2024

Introduction

This final assignment allows you to design and develop a Python-based project. The objective is to demonstrate your proficiency in problem-solving with Python scripting by applying the concepts covered throughout the course.

Assignment overview

You are required to:

- Design a modular Python program with functions and classes.
- Incorporate file input/output (I/O) operations.
- Utilize metaprogramming techniques (e.g., dynamic class modification, decorators).
- Implement custom error handling.
- Employ regular expressions where applicable.
- `It is mandatory to either push your project to GitHub with proper README.md files for installation instructions and setup, or submit your project via zip files.`
- Test your functions or classes using `pytest`. (Optional and for extra credit only)

In addition to the code, you will submit a comprehensive report explaining the structure, functionality, and design choices of your program. `Students are encouraged to follow the course description, but will not be strictly judged based on the number of words if they demonstrate extraordinary skills. The code itself will not count towards the final grade; however, you are expected to write a report explaining your code in detail.`

1 Part I: Meet Tarjan: The thoughtful planner

It's almost festival season in Seoul, a bustling metropolis where ten of Tarjan's beloved relatives reside, each scattered across different neighborhoods. Tarjan lives near Han River, in Yeouido, Seoul's financial district as he works in the financial sector. This year, Tarjan wants to make the festival extra special by personally bringing unique gifts to each relative. However, with limited time and the city's sprawling layout, Tarjan faces a significant challenge: how to efficiently visit all ten relatives without wasting precious time while navigating the busy streets.

Seoul is a big city and is connected by multi-ways through buses, trains, bicycles, and walking paths. The tentative location and transport mode with cost, speed, and time are given below. Your task is to develop a Python package named `TarjanPlanner` that computes the most efficient way for Tarjan.

Hints: To calculate the distance you can use either the Euclidean distance or the package `geopy` "from geopy.distance import geodesic". Some other useful libraries for this project could be

1. `networkx` for plotting the graph.
2. `matplotlib` for plotting plots.

Relatives, Street Name, District (Gu), Latitude, Longitude

Relative	Street Name	District (Gu)	Latitude	Longitude
Relative_1	Gangnam-daero	Gangnam-gu	37.4979	127.0276
Relative_2	Yangjae-daero	Seocho-gu	37.4833	127.0322
Relative_3	Sinsa-daero	Gangnam-gu	37.5172	127.0286
Relative_4	Apgujeong-ro	Gangnam-gu	37.5219	127.0411
Relative_5	Hannam-daero	Yongsan-gu	37.5340	127.0026
Relative_6	Seongsu-daero	Seongdong-gu	37.5443	127.0557
Relative_7	Cheongdam-ro	Gangnam-gu	37.5172	127.0391
Relative_8	Bukhan-ro	Jongno-gu	37.5800	126.9844
Relative_9	Samseong-ro	Gangnam-gu	37.5110	127.0590
Relative_10	Jamsil-ro	Songpa-gu	37.5133	127.1028

Table 1. Details of relatives, street names, districts, and geographic coordinates.

The network diagram showing the relative's position and modes of transport, find the best route for Tarjan is shown in Figure 1.

A. The report must include a flow chart that depicts how you designed the program.

B. Log the time required for your program to run.

C. Design an intuitive interface for users to input data, set preferences, and retrieve optimized routes.

D. Mention the limitations and considerations of your program.

Mode of Transport	Speed_kmh	Cost_per_km	Transfer_Time_min
Bus	40	2	5
Train	80	5	2
Bicycle	15	0	1
Walking	5	0	0

Table 2. Details of transport modes

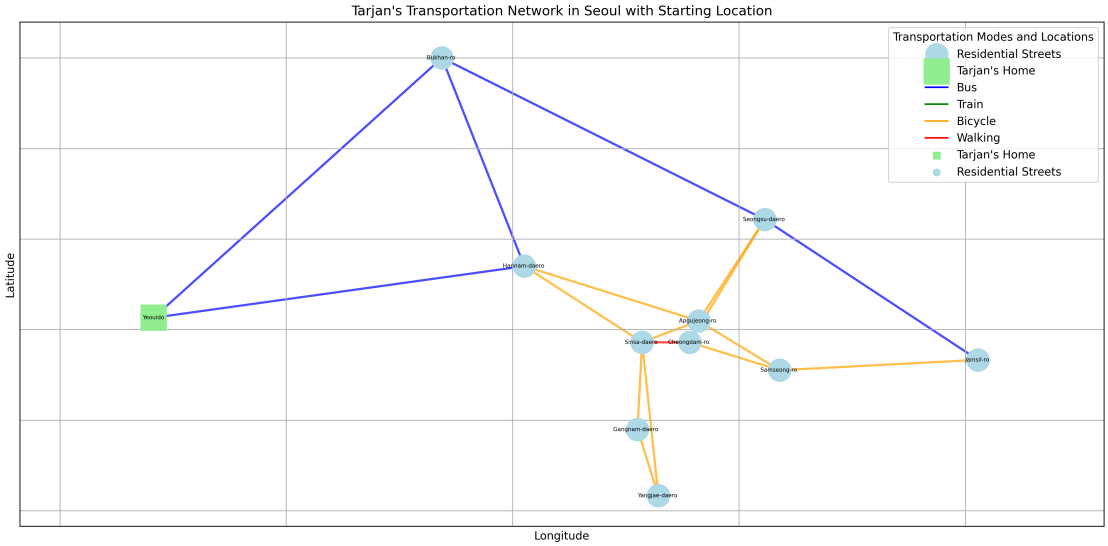

Figure 1. Network diagram showing the relative's position and modes of transport.

E. Allow the computation of routes based on different criteria, such as shortest travel time, least cost, or minimal number of transfers. (optional)

2 Part II: File organizer

Develop a script that organizes files in a directory based on their file types.

- **File handling**: Traverse directories and move files to appropriate folders.
- **Modules and packages**: Organize sorting logic, configuration, and logging into separate modules.
- **Metaprogramming**: Dynamically add new file type categories without modifying the core logic.
- **Error Handling**: Manage permissions issues and handle non-existent directories.
- **Regular expressions**: Identify file types using pattern matching.

3 Design and requirements

Your project should consist of multiple Python files/modules that work cohesively to solve your chosen problem. Below are the general guidelines for the project structure:

1. **Main module (`main.py`)**: The entry point of your application. It should orchestrate the program's functionality by calling appropriate functions or classes from other modules.
2. **Custom modules**: Implement at least three key functions or classes, each addressing a specific aspect of the problem. Organize these into separate files (e.g., `module1.py`, `module2.py`, etc.).
3. **File Handling**: Incorporate functionality that reads from or writes to files (e.g., configuration files, logs, data storage).
4. **Metaprogramming**: Apply metaprogramming techniques such as decorators or dynamic class modifications to enhance your code's flexibility and maintainability.

5. **Error Handling**: Implement custom error handling to manage runtime issues like file not found, invalid inputs, or network errors.

6. **Regular expressions**: Use regular expressions for tasks such as input validation, data parsing, or pattern matching where applicable.

7. **Testing**: Write unit tests for your functions or classes using `pytest`. Ensure that your tests cover various scenarios and edge cases. (optional)

4 Testing (Optional)

Implement unit tests to verify the correctness of your modules using `pytest`.

```
# test_project.py
import pytest
from module1 import Class1
from module2 import function2
from metaprogramming import DynamicClass

def test_load_data():
    obj = Class1()
    data = obj.load_data("test_data.txt")
    assert isinstance(data, str)

def test_function2():
    data = "Sample data"
    result = function2(data)
    assert result == "Expected Result"

def test_dynamic_method():
    obj = DynamicClass()
    result = obj.dynamic_method("Test")
    assert result == "Test"
```

Run the tests using the following command:

```
pytest test_project.py
```

5 Report guidelines

Your report should comprehensively document your project. Include (but not compulsion) the following sections:

- **Introduction:** Briefly explain the problem your program solves and its significance.
- **Design overview:** Describe the structure of your modules, classes, and functions. Explain how they interact to achieve the desired functionality.
- **Implementation:** Detail the implementation of key features, including file handling, metaprogramming techniques, error handling strategies, and the use of regular expressions.
- **Testing:** Explain your testing approach, including the tests you wrote and their outcomes. Include screenshots or logs of your `pytest` results. Optional

- **Conclusion:** Reflect on the challenges you faced, how you overcame them, and potential future improvements or extensions to your project.
- **References:** Cite any libraries, documentation, or resources you used during development.

A Appendix

A.1 Sample configuration files

Include any sample files that are part of your project, such as configuration files, data files, or templates.

A.2 Complete code

Provide snippets or GitHub repository links to your complete codebase if necessary.